
11. VC2

Program Name: VC2.java

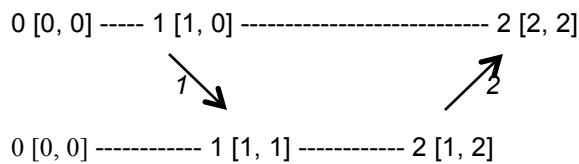
Input File: vc2.dat

Building off of the explanation of vector clocks in the previous problem, we can see how the vector clocks of events are updated in a distributed system as it completes its computation.

In this problem, we will represent the vector clocks as a sequence of numbers, separated by commas, in square brackets.

Say there is a new event on a process. If the event is not a receive event, then the only change that needs to be made is incrementing the event number for the current process it happens on. For example, if a non-receive event happens on the first process (0-indexed), and the last event that happened on that process had a vector clock of [2, 3, 4], this new event would have a vector clock of [3, 3, 4].

If the event is a receive, we have a slightly more complicated process. Let r be our receive event, s be the corresponding send event, and l be the previous event on this process. For each process i , $r[i] = \max(s[i], l[i])$, and afterwards, you must increment the event number of the local counter. For example, look at the following process execution of two processes communicating with a pair of messages:



In the above figure, process 0 sends a message 1 to process 1, and process 1 replies with 2. The format is $x \text{ VC}$, where x is the event number of that event, and VC is the vector clock of the event. We also give each message a message id. We assume each process has a “start” event of 0.

To look at an example, let's figure out the vector clock of event 2 on process 0. In this case, event 2 on process 0 is a receive event, and the corresponding send event, or “s”, is event 2 on process 1. Also, event 1 on process 0 is the previous event on process 0, or “l”. Thus, the vector clock of l is [1, 0], and the vector clock of s is [1, 2], and $r = [\max(s[0], l[0]), \max(s[1], l[1])] = [\max(1, 1), \max(0, 2)] = [1, 2]$. After, we must increment the local counter. Since the current event is happening on process 0, we must increment $r[0]$ by 1, from 1 to 2, so the final result is [2, 2].

Input

The first line will contain the number of test cases, T .

Each test case contains $P + N + 2$ lines. The first line is the number of processes P . The next P lines each contain every event in the event history of that process. Each process implicitly has a starting event 0, and every event listed will be either a send or receive. Thus each message will be SN or RN, where N is the unique message id of that message. The next line contains the number N , the number of vector clocks you are to compute. Each line after that will contain two integers p and k , meaning you need to find the vector clock of the k th event on the p th process.

Output

Each vector clock is enclosed by brackets, has the numbers separated by commas, with no spaces.

Constraints

$1 \leq T \leq 5$
 $0 \leq P \leq 10$
 $1 \leq N \leq 50$

Example Input File

```
2
2
S1 R2
R1 S2
1
0 2
2
S1 S3 R2 R4 S5
R1 S2 S4 R3 R5
2
1 1
0 5
```

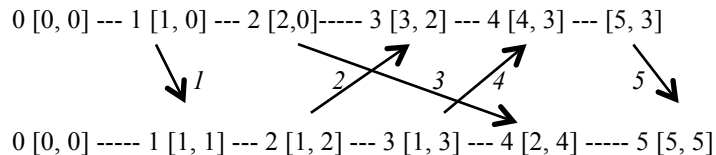
Example Output to Screen

```
[2, 2]
[1, 1]
[5, 3]
```

Explanation of Sample Output:

The first case is the example described above.

The second and third cases are illustrated below:



The mathematical explanation of the answers is as follows:

Event 1 on process 1 is a receive event, so we need to consider the previous local event (event 0 on process 1) and the corresponding send event, event 1 on process 0. Event 0 on process 1 has a vector clock of [0, 0], and event 1 on process 0 has a vector clock of [1, 0], so we apply the receive update rule, making the new vector clock, $[\max(1, 0), \max(0, 0)] = [1, 0]$, and then incrementing process 1's position, giving us [1, 1]

Event 5 on process 0 is a send event, so we only need to get the vector clock of the previous local event, event 4 on process 0, which has a vector clock of [4, 3], and we just need to increment position 0, resulting in [5, 3].

We can also sanity check these results with the definition provided in the VC1 outline of vector clocks, or the "last event on each process that happens-before this event".

For the first case, we are looking at event 2 on process 0. At this point, the vector clock at position 0 is obviously 2, since process 0 saw its own last event. Additionally, event 2 on process 1 was the sending of message 2, corresponding to the receive in event 2 on process 0, so that event is the latest event on process 1 to happen-before event 2 on process 0. Thus, our vector clock is [2, 2].

For the second case, event 1 on process 1, the last event it saw on process 0 was the send of message 1, event 1 on process 0, and the last event it saw on itself was this event, so the vector clock is [1, 1].

For the third case, event 5 on process 0, the last event on process 0 that process 0 has seen is 5, and the last event process 0 saw on process 1 was the send of message 4, which was event 3 on process 1, so the vector clock is [5, 3].